16 비트 EISC 마이크로 프로세서에 관한 연구

조 경 연[†]

요 약

8 비트와 16 비트 마이크로 프로세서는 소규모 제어기기에 많이 사용되고 있다. 이러한 실장 제어용 마이크로 프로세서는 CPU와 메모리 및 입출력회로가 하나의 반도체에 집적되어야 하므로 회로가 간단하고, 코드 밀도가 높은 것이 요구되고 있다.

본 논문에서는 코드 밀도가 높은 EISC(Extendable Instruction Set Computer) 구조를 가지는 16 비트 마이크로 프로세서인 SE1608을 제안한다. SE1608은 8개의 범용 레지스터를 가지며, 16 비트 고정 길이 명령어, 짧은 오프셋 인덱스 어드레싱과 짧은 상수 오퍼랜드 명령어를 가지며, 확장 레지스터와 확장 프래그를 사용하여 오프셋 및 상수 오퍼랜드를 확장할 수 있다.

SE1608은 FPGA로 구현하여 약 12,000 게이트가 소요되었으며, 8 MHz에서 모든 기능이 정상적으로 동작하는 것을 확인하였고, 크로스 어셈블러와 크로스 C/C++ 컴파일러 및 명령어 시뮬레이터를 설계하고 동작을 검증하였다.

SE1608의 코드 밀도는 16 비트 마이크로 프로세서인 H-8300의 140%, NM10200의 115%로 현격하게 높은 장점을 가진다. 따라서 하드웨어가 간단하고, 프로그램 메모리 크기가 작아지므로 실장 제어용 마이크로 프로세서에 적합하여 폭 넓은 활용이 기대된다.

A Study on 16 bit EISC Microprocessor

Gyeong-Yeon Cho[†]

ABSTRACT

8 bit and 16 bit microprocessors are widely used in the small sized control machine. The embedded microprocessor which is integrated on a single chip with the memory and I/O circuit must have simple hardware circuit and high code density.

This paper proposes a 16 bit high code density EISC(Extendable Instruction Set Computer) microprocessor. SE1608 has 8 general purpose registers and 16 bit fixed length instruction set which has the short length offset and small immediate operand. By using an extend register and extend flag, the offset and immediate operand in instruction could be extended.

SE1608 is implemented with 12,000 gate FPGA and all of its functions have been tested and verified at 8 MHz. And the cross assembler, the cross C/C++ compiler and the instruction simulator of the SE1608 have been designed and verified.

This paper also proves that the code density of SE1608 shows 140% and 115% higher code density than 16 bit microprocessor H-8300 and MN10200 respectively, which is much higher than traditional microprocessors. As a consequence, the SE1608 is suitable for the embedded microprocessor since it requires less program memory to any other ones, and simple hardware circuit.

1. 서 론

1970년대에 개발된 마이크로 프로세서는 제어 기기 분야 및 소형 컴퓨터에서 주로 사용되어 오다가 1980년대에 이르러 RISC(Reduced Instruction Set Computer)[1] 구조의 도입으로 중대형 컴퓨터에 이르기까지 광범위하게 사용되고 있다. 또한 반도체 기술의 급격한 발전으로 슈퍼스칼라 구조[2]가 마이크

† 정회원, 부경대학교 공과대학 전자 컴퓨터 정보통신 공학부 부교수

로 프로세서에도 적용되고 있으며 동작 속도도 수백 MHz에 이르고 있다[3-6].

한편 마이크로 프로세서는 실장 제어용으로 거의 모든 전자 제품 및 자동화 기기에서 채용하고 있다. 특히 냉장고, 에어콘, 전축, TV, 세탁기 등 가전기기와 Fax, 복사기, 프린터 등 사무용기기와 자동차, 선박, 자동화기계 등 사무 및 산업용 기기와 PDA (Personal Data Assistance), NC(Network Computer) 등 정보 기기 그리고 각종 오락기, 노래반주기 등 정보 기기 등에서 사용하는 실장 제어용 마이크로 프로세서 시장은 매년 10% 이상씩 성장하고 있으며, 21세기 산업을 주도하는 핵심 기술로 자리 매김하고 있다[7-8].

이러한 실장 제어용 기기는 마이크로 프로세서와 메모리 및 입출력 장치가 하나의 반도체에 집적되는 경우가 많다. 그런데 반도체 가격은 반도체 크기에 따라 결정되며, 가장 넓은 면적을 차지하는 것은 메모리이다. 따라서 반도체 가격을 낮추기 위해서는 메모리 크기를 줄여야 하며, 이를 위해서 코드 밀도가 높은 컴퓨터 구조에 대한 연구가 필요하다[8].

특히 8 비트 및 16 비트 마이크로 프로세서는 소규모 제어기기에서 대량으로 사용되어 전체 마이크로 프로세서 시장의 80%이상을 점유하고 있다. 그러나 8 비트 마이크로 프로세서는 워드 길이가 짧아서 고급언어 컴파일러 사용이 제한적이 되어 어셈블러를 사용해야 하는 단점을 가지며, 16 비트 마이크로 프로세서는 CISC(Complex Instruction Set Computer) 구조를 가지므로 하드웨어가 복잡하면서도 고급언어 컴파일러에 비효율적이다.

한편 최근에 오퍼랜드 길이를 확장하는 기능을 가지는 확장 명령어 세트 컴퓨터(Extendable Instruction Set Computer: EISC)에 대한 연구가 발표되었다. EISC는 특히 코드밀도가 높고, 하드웨어가 간단하고, 고급언어 컴파일러에 효율적인 장점을 가진다[9].

본 논문에서는 EISC 구조를 가지는 16 비트 마이크로 프로세서인 SE1608를 제안한다.

EISC는 RISC(Reduced Instruction Set Computer)와 유사한 명령어 특성을 가진다. 그러나 기존의 16 비트 RISC 마이크로 프로세서는 없으므로 32 비트 RISC 마이크로 프로세서인 MIPS-R3000으로 작성된 프로그램을 분석하여 명령어 사용 특성을 분석한다. C/C++ 컴파일러는 EGCS-1.1.2[10]을 사용하였으며 C 라이브러리 NEWLIB-1.8.1[11]과 C++ 라이브러리 LIBSTDC++-2.8.1[12]를 벤치마크 프로그램으로 하여 분석하였다. 이들은 C/C++로 작성된 대표적인 프로그램이고, 컴파일된 기계어 크기는 약 500 키로 바이트로 대상 프로그램으로 적합하다.

분석 결과 로드, 스토어 명령어가 차지하는 비중이 높으며, 짧은 길이의 오프셋을 사용하는 경우가 대부분임을 보인다. 또한 상수의 사용에서도 작은 크기의 상수 빈도가 높음을 보인다. 이러한 특성을 효율적으로 지원하기 위해서 16 비트 고정 길이 명령어를 가지며, 오프셋 및 상수 필드를 확장하는 명령어 세트를 설계한다. 또한 코드 밀도를 더욱 높이기 위하여 레지스터 리스트 푸시, 팝 명령을 가지며, 파이프라인 제어를 하드웨어로 수행하여 불필요한 NOP 명령어를 제거하였다.

제안한 SE1608은 FPGA로 구현하여 약 12,000 게이트가 소요되었다. 이것은 8 비트 마이크로 프로세서인 Z80이 10,000 게이트, I8051이 8,000 게이트가 소요된 것과 비교하면 SE1608의 하드웨어가 8 비트 마이크로 프로세서 수준으로 간단함을 알 수 있다. FPGA로 구현한 SE1608은 8 MHz에서 모든 기능이 정상적으로 동작하는 것을 확인하였다. 또한 크로스 어셈블러와 크로스 C/C++ 컴파일러 및 명령어 시뮬레이터를 설계하였다. 자료 구조 및 수학 함수 프로그램을 C/C++ 언어로 작성하고, 이들을 크로스 컴파일러와 어셈블러를 사용하여 기계어로 번역하고, 이것을 시뮬이터에서 동작시킨 결과와 IBM-PC에서 전용 컴파일러를 사용하여 얻은 결과를 비교하여 동작을 검증하였다.

그리고 제안한 SE1608을 기존의 16 비트 마이크로 프로세서와 코드 밀도를 비교하였다. 제안한 SE1608의 코드 밀도는 히다찌사의 H-8300의 140%, 파나소닉사의 MN10200의 115%로 기존 마이크로 프로세서에 비하여 현격하게 높으며 프로그램 크기도 이에 비례하여 작아 진다.

제 2장에서는 MIPS-R3000 프로그램을 분석하고 이에 따른 SE1608의 구조를 제안하며, 제 3장에서는 제안한 SE1608의 하드웨어 및 소프트웨어를 구현하고 제 4장에서는 8 비트 및 16 비트 마이크로 프로세서와 비교하여 성능을 검증하며, 제 5장에서는 결론을 맺는다. 그리고 부록에 제안한 SE1608의 명령어 세트를 보인다.

2. SE1608의 구조

2.1 8개 범용 레지스터

MIPS-R3000은 34개의 32 비트 레지스터를 가지고 있다. 2개는 곱셈 및 나눗셈 계산을 위한 전용 레지스터이며, 5개는 스택, 프레임 포인터, 조건 등을 저장하는 특수 레지스터로 사용하고 있어서 범용 레지스터로는 27개를 사용하고 있다. 표 1에는 EGCS C/C++ 컴파일러를 수정하여 MIPS-R3000의 범용 레지스터 수에 따른 컴파일러를 작성하고, 이를 이용하여 C 라이브러리 NEWLIB-1.8.1[11]과 C++ 라이브러리 LIBSTDC++-2.8.1[12]를 컴파일하여 생성한 기계어 프로그램의 특성을 표-1에 보인다.

표 1. MIPS-R3000에서 레지스터 수에 따른 프로그램 특성

No. of Register	Program size	Load/Store	Move
27	100.00	27.90%	22.58%
24	100.35	28.21%	22.31%
22	100.51	28.34%	22.27%
20	100.56	28.38%	22.24%
18	100.97	28.85%	21.93%
16	101.62	30.22%	20.47%
14	103.49	31.84%	19.28%
12	104.45	34.31%	16.39%
10	109.41	41.02%	10.96%
8	114.76	44.45%	8.46%

표 1에서 프로그램 크기는 범용 레지스터가 27개인 경우를 100으로 정규화한 수치로 나타내고 있다. 범용 레지스터 수가 작아지면 로드, 스토어의 빈도와 프로그램 크기가 증가한다. 로드, 스토어는 메모리 입출력을 동반하므로 데이터 전송 폭에 직접적인 영향을 미친다. 반면에 레지스터 수가 많아지면 명령어 길이가 길어져서 프로그램 크기가 커진다.

16 비트 마이크로 프로세서는 하드웨어가 간단해야 하므로 범용 레지스터 수를 가능한 작게 하는 것이 필수적이다. 한편 EGCS C/C++ 컴파일러에서는 최소한 8개의 범용 레지스터를 필요로 한다. 8개미만의 범용 레지스터를 가지는 경우에는 메모리 상에 레지스터를 설치해야 하므로 컴파일러 구성이 극히 비효율적이 된다.

이러한 점을 고려하여 본 논문에서 제안하는 SE1608에서는 8개의 범용 레지스터를 사용한다. 이후의 프로그램 분석은 8개 레지스터를 사용하도록 변형한 MIPS-R3000 컴파일러를 사용한다.

2.2 로드, 스토어 구조

표 2에 8개 범용 레지스터를 가지는 MIPS-R3000 프로그램에서 명령어 사용 빈도를 보인다.

표 2. 8개 범용 레지스터 MIPS-R3000에서 명령어 사용 빈도

Instruction	Frequency
move	8.46%
lw, sw	42.68%
nop	9.26%
addiu	6.91%
li	2.60%
lui	3.67%
sh, sb, lh, lb, lhu, lbu	1.77%
bnez, bne, beqz, beq, bltz, ...	5.95%
j, jal	9.31%
jr	1.56%
addu, subu, and, or, xor, nor, negu	2.81%
andi, ori, xori	1.91%
jalr	0.15%
slt, sltu, slti, sltiu	1.48%
sll, srl, sra, sllv, srlv, srav	1.20%
mult, multu, div, divu	0.08%
break, mfhi, mflo	0.11%

표 2에서 변수 산술연산 명령어(addu, subu, and 등)는 2.81%로 빈도가 높지 않다. 이들 명령어에서 메모리 변수를 사용하는 빈도는 출현 빈도보다 높지 않다. 즉 메모리 연산 명령어의 출현 빈도가 작으므로 이를 위한 명령어는 정의하지 않는 것이 효율적이다.

SE1608의 모든 연산 명령은 레지스터 오퍼랜드를 가지며, 메모리 입출력은 로드 스토어 명령어로 제한하는 로드, 스토어 구조를 가진다. 로드, 스토어 구조를 가지므로 하드웨어 구조가 단순해지고 따라서 동작 속도를 빠르게 할 수 있다.

2.3 16 비트 고정 길이 확장 명령어

표 2에서 'move' 명령어가 8.46%의 사용 빈도를 보이고 있다. SE1608의 범용 레지스터 수는 8개이므로 원시 레지스터 및 목적 레지스터 표현에 각각 3 비트가 필요하다. 따라서 'move' 명령어는 16 비트 길이 명령어로 표현할 수 있다.

16 비트 고정 길이 명령어를 사용하면 하드웨어가 단순하여 진다. 대부분의 명령어는 'move' 명령어와 같이 16 비트 고정 길이 명령어로 표현할 수 있으나, 로드, 스토어 명령과 상수 오퍼랜드 명령어는 오프셋 및 상수 오퍼랜드 길이를 감안하면 16 비트 고정 길이 명령어로 표현하기가 용이하지 않다.

표 2에서 워드 길이 로드, 스토어가 바이트 길이 로드, 스토어보다 높은 출현 빈도를 보이고 있다. 따라서 로드, 스토어 명령의 특성을 분석하기 위해서 표 3에 'lw(load word), sw(store word)'의 사용 특성을 보인다.

표 3. 'lw, sw' 명령어 특성

Offset length	Stack pointer (76.8%)	Index register (23.2%)
3 bit	43.5%	76.3%
4 bit	67.6%	80.4%
5 bit	85.9%	89.1%
6 bit	90.2%	91.4%
7 bit	97.2%	93.4%
8 bit	99.8%	99.9%

표 3에서 'lw, sw' 명령어의 76.8%가 스택 포인터를 사용하고 있다. 이것은 지역 변수 및 전달 변수가 스택에 설정되기 때문이다. 인덱스 레지스터를 사용하는 경우에는 3 비트 오프셋으로 77%의 명령어를 표현할 수 있다. 이것은 구조체의 크기가 크지 않은 것을 나타내며, 또한 자주 사용하는 변수를 구조체 앞단에 선언하는 것으로 그 빈도를 더욱 증가시킬 수 있다.

또한 상수 오퍼랜드 명령어인 'li(load immediate)' 명령어는 표 3에서 2.6%를 차지하며 상수의 출현 빈도는 표 4와 같다.

표 4로부터 'li' 명령어의 93.6%는 8 비트 상수로 표현할 수 있다.

표 4. 'li' 명령어에서 상수의 사용 빈도

Constant range	Frequency
-32 -- +31	72.4%
-64 -- +63	86.9%
-128 -- +127	93.6%
-256 -- +255	95.2%
others	100%

이상으로부터 'lw, sw' 명령어에서 짧은 길이 오프셋과 'li' 명령어에서 짧은 길이 상수 오퍼랜드 명령어의 출현 빈도가 높다는 것을 확인할 수 있다. 이러한 현상은 'addiu, slti, sltiu' 명령어 등에서도 공통적으로 나타난다.

이와 같이 빈도가 높은 짧은 길이 오프셋과 상수 오퍼랜드를 가지는 명령어를 16 비트 길이로 정의하면서, 긴 길이의 오프셋과 상수 오퍼랜드를 가지는 명령어도 16 비트 길이 명령어의 조합으로 표현하기 위해서 SE1608서는 16 비트 확장 레지스터(%ER)와 %ER 레지스터에 새로운 상수가 입력되었음을 나타내는 확장 프래그(E)를 설정한다. 또한 'LERI constant' 명령어를 다음과 같이 설정한다.

Instruction Mnemonics : LERI
Instruction Format : LERI constant
Instruction Representation :
  bit 15-12 = 0101
  bit 11-0 = Constant data bit 11-0
Operation :
  Load %ER with sign extend constant
  Set E flag

확장 프래그는 'LERI' 명령어를 수행하면 '1'이 되고, 오퍼랜드를 확장하는 명령어에서는 '0'이 된다. 로드, 스토어 명령어는 다음과 같이 정의한다.

Instruction Function : Load/Store
Instruction Representation :
  bit 15 = 0
  bit 14-12 = Operation
    000 : Sign extend 8 bit load.
      LDB (index, offset), dst
    001 : 16 bit load.
      LD (index, offset), dst

010 : 8 bit store.
STB src, (index, offset)
011 : 16 bit store.
ST src, (index, offset)
100 : Zero extend 8 bit load.
LDBU (index, offset), dst
bit 11 = Offset bit 5 if 8 bit load/store
Offset bit 6 if 16 bit load/store
bit 10-8 = Source/Destination register. %R0 thru %R7.
bit 7-5 = Index register. %R0 thru %R7.
bit 4-0 = Offset bit 4-0 if 8 bit load/store
Offset bit 5-1 if 16 bit load/store
Effective operand address : EA
If (E flag is 0)
EA = Zero extend offset + Index register
If (E flag is 1)
EA = %ER << 4 + Offset + Index register

이와 같이 확장 프래그에 의하여 긴 오프셋과 긴 상수 오퍼랜드로 확장하는 구조를 채용하면 모든 명령어를 16 비트 고정 길이로 표현하는 것이 가능하다.

2.4 스택 명령어

표 3에서 'lw, sw' 명령어의 오프셋 길이가 스택 포인터와 인덱스 레지스터를 사용하는 경우에 현격한 차이를 보이고 있다. 표 3으로부터 스택 포인터를 사용하는 경우에 'lw, sw' 오프셋은 5 비트 이상이 필요하다. SE1608에서는 이러한 특성을 감안하여 8 비트 오프셋을 가지는 스택 포인터 'lw, sw' 명령어를 별도로 정의한다.

또한 'lw, sw' 명령어에서 스택에 푸시 팝하는 빈도가 15.8%로 조사되었으며 8개 레지스터를 하나의 리스트로 선언하면 평균 푸시 팝 레지스터 수는 4.3개로 조사되었다. 그러므로 SE1608에서는 레지스터를 하나의 리스트로 묶어서 표현하는 푸시 팝 명령어를 정의한다. 이러한 푸시 팝 레지스터 리스트는 여러 개의 메모리 동작을 발생시키는 명령어로 RISC 구조에서는 사용되지 않으나 CISC 구조에서는 많이 사용하고 있다.

표 2에서 'addiu(add immediate)' 명령어의 출현 빈도는 6.91%이다. 이중에서 스택 포인터를 사용하는 빈도는 42%이며, 이중 8 비트 상수의 빈도가 99.9%이다. 그러므로 SE1608에서는 9 비트 상수를 스택 포인터에 더하고, 빼는 명령어를 정의한다.

2.5 기타 명령어

표 2에서 조건 분기 명령어의 빈도가 5.95%이다. 이를 효율적으로 지원하기 위해서 SE1608에서는 캐리, 사인, 제로, 오버플로우의 4개 플래그를 사용하며 이들을 조합하여 14가지 조건 분기 명령어를 만든다. 조건 분기 명령어의 오프셋은 9 비트로 설정하며, 확장 레지스터를 이용하여 16 비트까지 확장한다.

SE1608의 범용 레지스터가 8개로 레지스터 지정에 3 비트의 명령어 비트가 소요되어 3 오퍼랜드 명령어를 구성할 수 있다. 따라서 SE1608의 논리 산술 연산 명령어는 3 오퍼랜드 형식을 가진다.

곱셈 명령어는 출현 빈도는 낮으나 멀티미디어 등 특정 응용 분야에서 사용 빈도가 특히 높으며, 구현 방식에 따라 동작 속도에 차이가 크다. 3 오퍼랜드 곱셈 명령어를 설정한다.

3. 하드웨어 및 소프트웨어 구현

표 5에 SE1608의 레지스터 구성을 보인다.

표 5. SE1608의 레지스터 구성

Register name	Description
R0 - R7	16 bit general purpose register
PC	16 bit program counter
SP	16 bit stack pointer
BK	16 bit break point register bit 15-1 = Break address bit 0 = Break enable
ER	16 bit extension register
SR	16 bit status register bit 15 = Reserved bit 14 = Enable NMI bit 13 = Enable maskable interrupt bit 12 = Auto vectored if 0 bit 11 = Extend flag (E) bit 10-8 = Reserved bit 7 = Carry flag (CY) bit 6 = Zero flag (Z) bit 5 = Sign flag (S) bit 4 = Overflow flag (V) bit 3-0 = Reserved

프로그램상의 오류를 찾아내기 위해서 실행 정지 레지스터(break point register)를 설정하였다. 실행 정지 레지스터에 설정된 번지와 프로그램 카운터가 일치하면 실제 명령어와 상관없이 실행 정지 명령어가 입력된다. SE1608에서는 실행 정지 명령어로 소프트웨어 인터럽트 백터 4를 사용하였다. 한편 SE1608은 16 비트 고정 길이 명령어이므로 프로그램 어드레스의 비트 0는 사용하지 않는다. 이점을 이용하여 실행 정지 레지스터의 비트 0를 실행 정지 가능 비트로 할당하였다.

파이프라인은 메모리 읽기, 쓰기를 수반하지 않는 명령어는 2 스테이즈로 동작하고, 읽기, 쓰기를 수반하는 명령어는 메모리 접근 스테이지를 추가하는 구조로 설계하였다. 따라서 읽기, 쓰기를 수반하지 않는 명령어의 수행에는 한 클럭이 소요되며, 읽기, 쓰기를 수반하는 명령어는 메모리 접근 수보다 하나 많은 클럭이 소요된다.

제안한 SE1608은 FPGA로 구현하여 약 12,000 게이트가 소요되었으며, 8 MHz에서 정상적으로 동작하는 것을 확인하였다. 조합 논리회로 곱셈기와 바렐 쉬프터를 채용하였다. 설계한 SE1608은 LED 제어기, RS-232C 제어기, 타이머, 인터럽트 제어기, 메모리 제어기를 갖춘 FPGA와 함께 인쇄 회로 기판에 장착하였고, RS-232C로 IBM-PC와 연결하여 프로그램을 다운 로드 받아서 수행하고, 수행 결과를 LED에 표시하거나 RS-232C를 통하여 IBM-PC에 출력하여 동작을 검증하였다.

표 6에는 크로스 소프트웨어 구현 현황을 보인다.

표 6. SE1608 크로스 소프트웨어

Host platform	Window-95, Window-NT, Linux, SUN
Object file format	ELF
Cross assembler	FSF Binutils-2.9.1
Cross loader	FSF Binutils-2.9.1
Binary utilities	FSF Binutils-2.9.1
Cross C compiler	Cygnus EGCS-1.1.2
Cross C++ compiler	Cygnus EGCS-1.1.2
C library	Cygnus NEWLIB-1.8.1
C++ library	FSF LIBSTDC++-2.8.1
Cross debugger	FSF GDB-4.18
Simulator	FSF GDB-4.18
Real Time OS	uC/OS-1.5

크로스 소프트웨어는 프리웨어로 제공되는 프로그램을 포팅하여 개발하였으며, 개발의 편의성을 위하여 IBM-PC의 Linux OS 상에서 개발하였고, 개발된 소프트웨어를 Window와 SUN에 재포팅하였다. 수학 함수 및 자료 구조 등에 관한 예제 프로그램을 C와 C++로 작성하고 크로스 소프트웨어로 컴파일하여 기계어를 생성하고, 생성된 기계어를 시뮬레이터에서 수행하여 얻은 결과를 IBM-PC의 전용 컴파일러에 의한 결과와 비교하여 동작을 검증하였다.

4. 성능 평가

코드 밀도는 프로그램 크기에 반비례한다. 즉, 코드 밀도가 높으면 프로그램 크기가 작아진다. 본 논문에서는 상대 프로그램 크기(Relative Program Size : RPS)를 대상 마이크로 프로세서에 대한 SE1608의 상대 코드 밀도의 역수로 식 (1)과 같이 정의한다.

$$RPS = \frac{Code-Density-of-SE1608}{Code-Density-of-Object-Microprocessor} = \frac{Program-Size-of-Object-Microprocessor}{Program-Size-of-SE1608} \quad (1)$$

성능 평가를 위하여 표 6의 환경으로 16 비트 마이크로 프로세서인 히다찌사의 H-8300과 파나소닉사의 MN10200에 대한 크로스 C/C++ 컴파일러를 작성하였다. 객관적인 성능 평가를 위해서 C 라이브러리 NEWLIB-1.8.1[11]과 C++ 라이브러리 LIBSTDC++-2.8.1[12]를 벤치마크 프로그램으로 선정하였다. NEWLIB-1.8.1은 Cygnus 사에서 실장 제어용으로 개발한 C 라이브러리로 ANSI C 라이브러리와 SUN 사에서 SunPro 워크스테이션에서 사용하고 있는 단정도 및 배정도 실수 함수 라이브러리로 구성되어 있다. LIBSTDC++는 SGI 사의 STL(C++ Standard Template Library)로 C++ 입출력 함수와 수학 함수 라이브러리이다. 이들은 C/C++로 작성된 대표적인 프로그램이고, 컴파일된 기계어 크기는 MIPS-R3000에서 381,560 바이트이며, 기계어 구성은 표 2와 같다.

표 7에 벤치마크 프로그램을 사용하여 제안한

SE1608에 대한 기존 마이크로 프로세서들의 상대 프로그램 크기 및 특징을 비교한다.

표 7. 16 비트 마이크로 프로세서의 비교

Processor	Relative Program Size	No. of GPR	Instruction Length	Hardware
SE1608	1.00	8	Fixed 16 bit	Simple
H-8300	1.40	7	Variable	Complex
MN10200	1.15	3/4	Variable	Complex

표 7에서 현재 16 비트 마이크로 프로세서로 가장 많이 사용되고 있는 히다찌사의 H-8300의 상대 프로그램 크기가 1.40이다. 즉, H-8300의 프로그램 크기가 SE1608보다 40% 크다. 또한 파나소닉사의 MNH10200의 상대 프로그램 크기는 1.15이다. 따라서 MN10200의 프로그램 크기는 SE1608보다 15% 커진다.

MN10200은 어드레스 레지스터와 데이터 레지스터로 구분되어 각각 4개씩을 가지고 있다. H-8300은 7개의 레지스터와 1개의 스택 포인터를 가지고 있다. 반면에 SE1608은 8개의 범용 레지스터와 별도의 스택 포인터를 가진다. 또한 SE1608은 스택 포인터의 어드레싱 오프셋을 8 비트로 하여서 C/C++ 등 고급언어에서 지역변수를 효율적으로 지원하고 있으므로 고급언어 컴파일러에 효율적이다.

또한 H-8300과 MN10200은 다중길이 명령어를 가지는 CISC 구조이므로 하드웨어가 복잡한 반면에 SE1608은 16 비트 고정 길이 명령어를 가지므로 하드웨어가 이들보다 간단하여 진다.

표 8에 대표적인 8 비트 마이크로 프로세서인 Z80[14]과 I8051[15]과의 비교를 보이고 있다.

표 8. 8 비트 마이크로 프로세서의 비교

Processor	Gate count	Multiplier	Shift	Compiler
SE1608	12,000	16 bit H/W	Barrel	Efficient
SE1608-1	10,000	S/W	Barrel	Efficient
Z80	10,000	S/W	1 bit	Inefficient
I8051	8,000	8 bit H/W	1 bit	Inefficient

SE1608-1은 SE1608에서 16 비트 하드웨어 곱셈기를 제외한 모델이다. 표 8로부터 SE1608의 하드웨어는 8 비트 마이크로 프로세서와 비슷한 정도로 간단함을 알 수 있다. 특히 16 비트 하드웨어 곱셈기를 제외하면 Z80과 동일한 게이트로 구성할 수 있는 데, SE1608이 바렐 시프트를 더 가지고 있는 점을 고려하면 실제적으로는 Z80보다 간단하다.

5. 결 론

마이크로 프로세서는 실장 제어 기기 분야에서 중대형 및 소형 컴퓨터에 이르기까지 광범위하게 사용되고 있으며, 반도체 기술의 급격한 발전으로 실장 제어 분야에서는 8 비트 또는 16 비트 마이크로 프로세서와 메모리 및 입출력 장치가 하나의 반도체에 집적되는 기술이 보편화되고 있다. 이러한 실장 제어 기기에서는 반도체 가격을 낮추기 위해서는 메모리 크기를 줄여야 하며, 이를 위해서 코드 밀도가 높으면서도 하드웨어가 간단한 8 비트 또는 16 비트 마이크로 프로세서 구조에 대한 연구가 필요하다.

이러한 필요성에 따라 본 논문에서는 코드밀도가 높으면서 하드웨어 구조가 간단한 16 비트 확장 명령어 세트 컴퓨터(Extendable Instruction Set Computer: EISC) 마이크로 프로세서인 SE1608을 제안하였다.

제안한 SE1608은 8개의 16 비트 범용 레지스터, 로드, 스토어 명령어 형식, 16 비트 고정 길이 명령어, 짧은 길이 오프셋 어드레싱, 짧은 길이 상수 오퍼랜드 명령어를 가지며 확장 레지스터와 확장 프래그를 채용하여 오프셋 및 상수 오퍼랜드를 확장할 수 있다.

제안한 SE1608은 FPGA 회로로 구현하여 약 12,000 게이트가 소요되었으며, 8 MHz에서 모든 기능이 정상적으로 동작하는 것을 확인하였고, 크로스 어셈블러와 크로스 C/C++ 컴파일러 및 명령어 시뮬레이터를 설계하였다. 자료 구조 및 수학 함수 프로그램을 C/C++ 언어로 작성하고, 이들을 크로스 컴파일러와 어셈블러를 사용하여 기계어로 번역하고, 이것을 시뮬이터에서 동작시킨 결과와 IBM-PC에서 전용 컴파일러를 사용하여 얻은 결과를 비교하여 동작을 검증하였다.

제안한 SE1608의 코드 밀도는 16 비트 마이크로 프로세서로 널리 사용되고 있는 히다찌사의 H-8300의 140%, 파나소닉사의 MN10200의 115%로 높은 장점을 가진다. 또한 하드웨어는 8 비트 마이크로 프로세서인 Z80과 I8051과 비슷한 정도로 간단하다. 그리

고 EISC 구조를 가지고 있으므로 C/C++ 등 고급언어 컴파일러에 효율적이다.

따라서 프로그램 메모리 크기가 작아지면서, 하드웨어가 간단하고, 고급언어 사용이 효율적이므로 실장 제어용 마이크로 프로세서에 특히 적합하여 폭넓은 활용이 기대된다.

참 고 문 헌

[1] D. Patterson, "Reduced Instruction Set Computer," Comm. ACM, Vol. 28, No. 1, pp. 8-21, Jan. 1985

[2] Dezso Sima *et al.*, "Superscalar Instruction Issue," IEEE Micro, pp. 28-39, Oct. 1997

[3] B. Gieseke *et al.*, "A 600MHz Superscalar RISC Microprocessor with out-of-order execution," ISSCC Digest Tech. Papers, pp. 176-177, Feb. 1997

[4] C. A. Maier *et al.*, "A 533MHz BiCMOS Superscalar RISC Microprocessor," IEEE Journal of Solid-State Circuits, Vol. 32, No. 11, pp. 1625-1634, Nov. 1997

[5] Charles F. Webb *et al.*, "A 400MHz S/390 Microprocessor," IEEE Journal of Solid-State Circuits, Vol. 32, No. 11, pp. 1665-1675, Nov, 1997

[6] Paul E. Gronowski *et al.*, "High-Performance Microprocessor Design," IEEE Journal of Solid-State Circuits, Vol. 33, No. 5, pp. 676-686, May 1998

[7] Doug Burger, "Limited Bandwidth to Affect Processor Design," IEEE Micro, pp. 55-62, Dec. 1997

[8] Manfred Schlett, "Trends in Embedded-Microprocessor Design," IEEE Computer, pp. 44-50, Aug. 1998

[9] 조 경연, "확장 명령어 32 비트 마이크로 프로세서에 관한 연구," 전자공학회논문지, 제 36권 D편 제 5호, pp. 11-20, May 1999

[10] ftp://sourceware.cygnus.com/pub/egcs/released/egcs-1.1.2.tar.gz

[11] ftp://sourceware.cygnus.com/pub/newlib/released/newlib-1.8.1.tar.gz

[12] ftp://cair-archive.kaist.ac.kr/pub/gnu/released/libstdc++-2.8.1.tar.gz

[13] ftp://sourceware.cygnus.com/pub/gdb/released/gdb-4.18.tar.gz

[14] Z80 Family Databook, Zilog, 1994

[15] Embedded Controller Handbook, Intel, 1988

부록 : SE1608 명령어 세트

Type 0 : Load/store
- bit 15 = 0
- bit 14-12 = Operation
- bit 11 = Offset
- bit 10-8 = Source/Destination register
- bit 7-5 = Index register
- bit 4-0 = Offset

Type 5 : Load extension register and set E
- bit 15-12 = 0101
- bit 11-0 = Immediate data bit 11-0

Type 8 : Stack area 16 bit load/store
- bit 15-12 = 1000
- bit 11 = Operation
- bit 10-8 = Source/Destination register
- bit 7-0 = Offset bit 8-1

Type 9 : Push/Pop register list
- bit 15-12 = 1001
- bit 11 = Operation
- bit 10-0 = Register

Type 10 : Arithmetic/Logic immediate operation
- bit 15-13 = 101
- bit 12-9 = Immediate data
- bit 8-6 = Operation
- bit 5-3 = Source register
- bit 2-0 = Destination register

Type 12 : Arithmetic/Logic register operation
- bit 15-12 = 1100
- bit 11-9 = Source 2 register
- bit 8-6 = Operation

bit 5-3 = Source 1 register
bit 2-0 = Destination register

Type 13 : Conditional branch and CALL
bit 15-12 = 1101
bit 11-8 = Operation
bit 7-0 = Offset bit 8-1

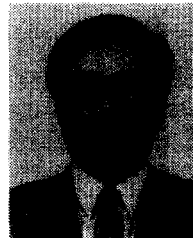
Type 14-0 : Misc instructions
bit 15-12 = 1110
bit 11-8 = 0000
bit 7-4 = Operation
0000 : 8 bit sign extend
0010 : Load BK
1000 : Register indirect JMP
1001 : Register indirect CALL
1010 : Set status flag 15 to 0
1011 : Clear status flag 15 to 0
1100 : Software Interrupt
1101 : Halt
1111 : Break point for debugger
bit 3-0 = Operand

Type 14-2 : Add stack pointer with offset
bit 15-12 = 1110
bit 11-8 = 0010
bit 7-0 = Offset bit 8-1

Type 14-4 : Signed integer multiply
bit 15-12 = 1110
bit 11-9 = 010
bit 8-6 = Source 2 register
bit 5-3 = Source 1 register
bit 2-0 = Destination register

Type 14-8 : Load immediate data
bit 15-12 = 1110
bit 11 = 1
bit 10-8 = Destination register
bit 7-0 = Immediate data

Type 15-0 : Static shift
bit 15-12 = 1111
bit 11-9 = 000
bit 8-5 = Immediate count
bit 4-3 = Operation
bit 2-0 = Destination register

Type 15-2 : Dynamic shift
bit 15-12 = 1111
bit 11-8 = 0010
bit 7-5 = Count register
bit 4-3 = Operation
bit 2-0 = Destination register

조 경 연

1990년 인하대학교 공과대학 전자공학과 정보공학전공 박사
1983~1991 삼보컴퓨터 기술연구소 책임연구원
1991~현재 부경대학교 전자 컴퓨터 정보통신 공학부 부교수
1991~현재 삼보컴퓨터 기술연구소 비상임 기술고문
1998~현재 아시아 디자인(주) 비상임 기술고문
관심분야 : 전자계산기구조, 반도체 회로 설계